Resilient Data Replication Mechanisms for Battle Command Systems

Samuel C. Chamberlain, Ph.D.
U.S. Army Research Laboratory
Aberdeen Proving Ground, Maryland, 21005–5067

ABSTRACT

As the move to automate the information distribution process progresses, commercial distributed database practices have become the focus of many system designs. However, at the lower echelons, where wireless communications are the rule, this has not been the case because (1) the problem is still viewed as a sophisticated, e–mail challenge to be solved using procedural messages rather than databases and (2) the rigidity of commercial practices does not support an environment that is characterized by the unfortunate combination of time–critical distribution requirements and tenuous communications. This paper describes the concepts and rationale behind *resilient,* as opposed to *robust,* data replication and introduces a "promiscuous" mechanism that addresses the requirement for highly selective information distribution when bandwidth is tenuous. The final conclusion is not surprising: "There is no free lunch."

1 Background

Last year, a paper was presented at this symposium that described a different perspective towards building battle command systems, a perspective that focuses on truly exploiting available computational power through *model-based, computationally intensive* paradigms of battle command rather than the current *communications–intensive, message–based* approach [Chamberlain, 1995]. Using this approach, a formal data model of the battlefield serves as the hub of information flow. This is in contrast to the traditional message–based approach where the data model and its container, the database, are viewed as only supporting entities. Under the model–based paradigm, each node maintains its own independent database, reflecting the battlefield situation to the best of its ability. From this vantage point, the task of communications switches from exchanging messages to updating each other's databases. Thus, the database provides the conduit, as well as the hub, by which information is transferred between different units and, often, between the applications within the same unit (or system).

The preceding description is easily recognized as that of a distributed database system that is typically implemented above a distributed computing environment. One impetus for development of the model–based approach is to make a true distributed computing environment viable for the lowest echelons (e.g., on platforms such a tanks, aircraft, or individual warriors) where communication links are often tenuous at best. At these echelons, automating battle command is further complicated by the real–time nature of many of the tasks (e.g., situational awareness, targeting, etc.). This paper describes the concepts and rationale behind *resilient,* as opposed to *robust,* data replications and "promiscuous" mechanisms are introduced to implement such a scheme.

2 Associated Technologies and Issues

2.1 Active Databases and Triggers

To implement a model–based paradigm, active database techniques may be used (see McCarthy and Dayal [1989], Cohen [1989], Hansen and Widom [1992], and Dayal *et al.* [1995]). In an active data-

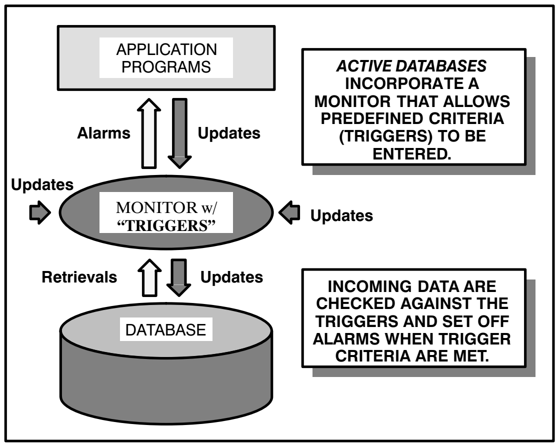

APPLICATION PROGRAMS
Alarms
Updates
Updates
MONITOR w/ "TRIGGERS"
Updates
Retrievals
Updates
DATABASE
ACTIVE DATABASES INCORPORATE A MONITOR THAT ALLOWS PREDEFINED CRITERIA (TRIGGERS) TO BE ENTERED.
INCOMING DATA ARE CHECKED AGAINST THE TRIGGERS AND SET OFF ALARMS WHEN TRIGGER CRITERIA ARE MET.

Figure 1: Active Database Structure

base incoming data are compared with a set of predefined queries called ‘‘triggers.’’ When a trigger fires, an associated action is executed. Typically, the action is to notify an application program with an alarm, see Figure 1. However, any action may be invoked by the trigger to include the replication of data. Thus, active database triggers can be used to control the flow of information, or in other words, to control synchronization between replicated or distributed databases, see Figure 2.

2.2 Context–Sensitive Information Distribution

When one asks a military end user about system requirements, the response is often: ‘‘Well, it depends upon the situation.’’ A major advantage of active databases is that *any* information that is maintained in the database may be used in the trigger criteria. Since each node may now maintain its own independent model of the battlefield, this provides a general–purpose monitoring system for any application or process. For example, if the current state of the connected communications system is maintained in the database, then that information can be used in the trigger criteria. Average network delay is often of keen interest to users. If it is maintained in the database, then it may be referred to by the trigger criteria to control whatever action is associated with the trigger. Therefore, as delay varies, so can the information that is exchanged. Similarly, unit status information may be used. The fact that a unit is ‘‘in contact’’ may cause a different set of triggers to be fired by an event than when the unit is in an Assembly Area. Both of these examples represent just another attribute in the database. No new software needs to be created to handle these cases. This approach facilitates many new avenues for automation because it *easily* allows the system to react ‘‘to the tactical situation.” Therefore, information distribution may be context sensitive where the context is the tactical situation that is represented in the database.

2.3 Consistency versus Synchronization

A major issue of data replication is data consistency (i.e., the property that the same fact in two different databases contains the same value). Two good references on this topic in the context of battle command are Davis and Ginn [1995] and Kameny [1995].

In commercial data replication mechanisms, the concept of data integrity, enforced via the audit trail, is paramount. Historically, in a distributed database system, tight–consistency is employed. That is, one is not allowed to execute an update until everyone (in the group) is ready to update his/her database too. This is not unreasonable for most applications (e.g., we want a consistent view of our bank statement or our airline seat assignment). Even in

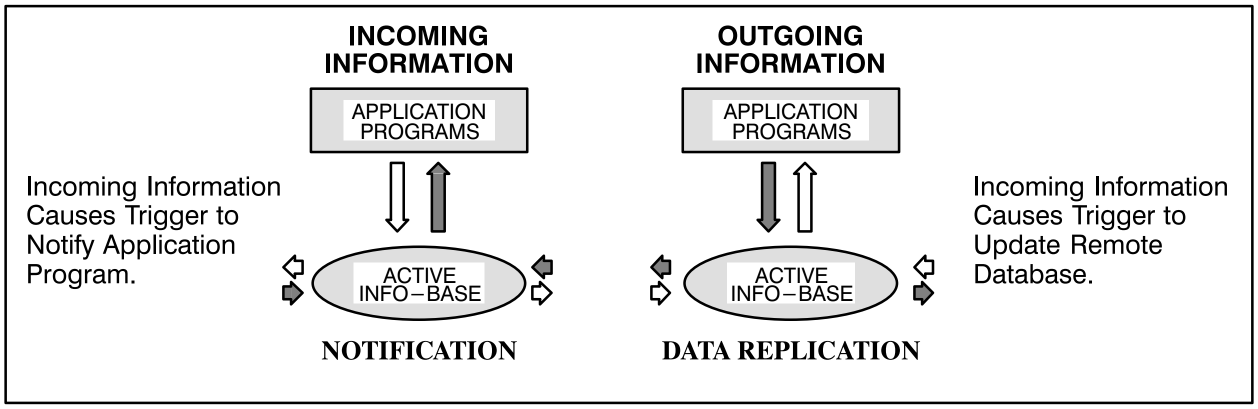


Figure 2: Active Databases Can Be Used to Control Synchronization Thresholds Between Databases

newer ‘‘loose–replication’’ mechanisms, integrity via the audit trail is still paramount. If connectivity to another database is lost, the transaction is allowed to occur locally while the remote transaction(s) are cached (i.e., queued) for later distribution. When the connection is resumed, a first–come, first–served approach is used until all the transactions are applied to the receiving databases. Thus, a common audit trail (or history) is maintained at each node with only the transaction times being different (i.e., the number and order of transactions is consistent).

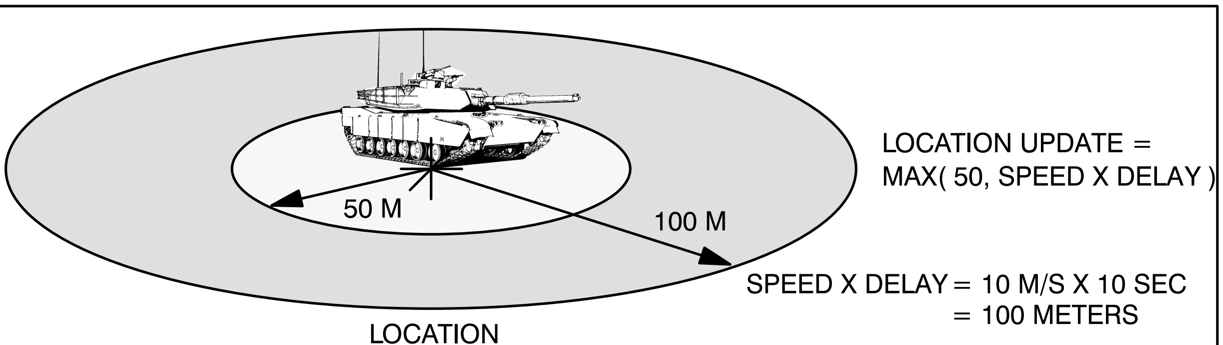
Consider what this might mean to some battle command applications. Suppose units are updating their position locations once a minute and the communications link is severed for ten minutes. Once communications are restored, the ten-minute-old updates are sent, followed by the nine-minute–old updates, and so forth, until all the updates are passed. In reality, all the commander may need is the most recent location update.

In another example, ground commanders typically maintain information at a resolution two echelons below theirs. This means that battalion commanders receive information about their platoon–sized echelons. Imagine more than a dozen of these databases frequently trying to update the battalion commander’s database via an intermediate node (their company commanders). This is simply unrealistic given the communications performance available at these echelons. For these reason, a new style of replication mechanism is required.

In many military situations, timing is more important than audit trail. A small amount of current data is usually more valuable than large amounts of old, but consistent data. The meaning of ‘‘old, but consistent’’ is that the data were correct at an earlier time (e.g., my location at time T is XY) but is not the most recent value (the correct but old data are referred to here as ‘‘stale’’ information). However, some amount of staleness is OK, especially if the approximate time lag is known. Because timing is the driving factor, data *synchronization* may be a better criterion than consistency because it implies a predefined or known rate of interaction.

At the low echelons, real–time data may be frequently entered into a local database from several on–board systems (e.g., global positioning system [GPS] receivers, inertial reference systems, speed sensors, logistics information, etc.). Although the latest data are maintained locally, this is an unrealistic expectation for remote databases (sites) due to severe communication constraints. Instead, some level of data synchronization will be required. Often, this is defined isochronally (e.g., updates every 30 seconds or every 100 meters), but in reality, it depends on the situation.

Fortunately, this is exactly the type of task for which active databases are designed. By linking the data replication mechanism to the triggers, any information that resides in the database can be used to define situation–dependent interdatabase synchronization criteria. In other words, triggers can be used to describe how far out of synchronization the databases (located at the different nodes) can become without significantly affecting current military operations. In Figure 3, even though the local database may be receiving updates every two seconds from the on–board GPS, position updates to other sites are based upon a realistic, desired rate or



A Ground Speed Of 10 m/s (22.5 mph) With An Average Network Delay Of 10 seconds Means An Update Every 100 meters At Best; Otherwise, One Will Just Saturate The Output Queue.

Figure 3: Location Synchronization Varies With Conditions (Vehicle Speed & Network Delay)

the best possible rate given the current network conditions (e.g., every 50 meters or the average vehicle speed times the average network delay). In the example given, network performance is the controlling parameter. Position updates arriving more often than every 100 meters will simply sit in the queue because they are arriving at a rate faster than the network can handle.

2.4 Robustness versus Resilience

One often hears the term ''robust'' used when describing communication or data replication systems. However, robust is defined as ''exhibiting strength,'' which implies a system that has significant force and power to fend off and repel attacks. The word ''resilient,'' on the other hand, refers to something that is ''stressed or deformed without permanently harming it.'' At the lower echelons, this is often the best we can do; that is, to ''take a few punches'' and rebound after the attack. This is especially true if the ''punches'' are not really hurting us. In battle command terms, this would mean not significantly disrupting our synchronization. Thus, we talk of *promiscuous* data replication mechanisms that, when the situation warrants, willfully allow the databases to drift out of synchronization, but within known limits.[1]

3 Promiscuous Data Replication

At the lower military echelons, communications are often tenuous at best. Average data rates can be relatively very low (hundreds of bits per second) with delays of tens of seconds and connectively can be notoriously intermittent. Even loose–replication mechanisms can become ''bogged down'' in this environment where update failures can exceed 50%. However, it is surmised that this situation can be handled effectively if we consider *tactically significant consistency* as our goal. In other words, the question should be asked: Just how consistent do we really need to be to still accomplish our task? The approach to achieve tactically significant consistency is to, one, be selective in what is sent, two, be efficient in sending information with special care not to waste bandwidth (i.e., saturate queues) with outdated information, and three, replace missing information with predictions based upon *a priori* information stored in the local database.

3.1 Distribution Rule

As previously stated, an active database trigger can have any action associated with it. The most common action is to *notify* an application program with an alarm. Similarly, a replication command can be invoked by the trigger. For this paper, a trigger with a replication command is called a *distribution rule*, or just *rule*.

In one prototype implementation (by the U.S. Army Research Laboratory, ARL), distribution rules have the general form

IF *criteria* THEN *action1* ELSE *action2*,

where the IF portion is the trigger mechanism, the THEN portion invokes the replication mechanism, and the ELSE portion invokes the failure resolution action. The trigger criteria are tested each time new information is received by the database (a typical example is position location data that may arrive every two seconds from an on–board GPS). The trigger criteria can include any mixture of current database information, the newly arrived information (that caused the trigger to fire), mathematical expressions and constants, and meta–information such as the source or destination of the information, on what network it arrived, or whether it is new or updated information. Note that at the low echelons, broadcast radio networks are the norm; therefore, overhearing of information exchanged between other sites is also passed on to the active database.

The replication mechanism contains information about the what, where, and how information should be sent to another database. These will be referred to as the ''what–part,'' ''where–part,'' and ''how–part,'' respectively. As with the trigger mechanism, any information that is stored in the database is accessible by the replication mechanism to instantiate these parameters. Thus, dynamic database values (i.e., variables) as well as constants can be used in the rules. So the first objective (to be selective in what is sent) is accomplished in two ways: first, active database triggers identify the tactically significant events that warrant the exchange of information, and second, the ''what part'' of the replication action determines what information is to be exchanged as a result of an event.

1 From Webster 7th Collegiate Dictionary:
Robust – Exhibiting strength.
Resilient – Capable of withstanding shock without permanent deformation or rupture.
Promiscuous – Casual, irregular.

3.2 Fact Exchanges

As with any distributed database system, the information exchanged between the databases is in the form of individual database transactions (usually updates). This is in contrast to a message–based approach where a predefined message that is composed of a standard set of fields is exchanged; see VMF [1996]. The advantage of the message–based approach is terseness because fixed sets of fields are predefined, thus eliminating most of the field identifiers. However, the cost is inflexibility of content and the added complexity required to build parsers to map message fields into database fields. Adding new fields to a message is a huge bureaucratic undertaking. In the database approach, the content of a data exchange (analogous to a "message") is determined by the rules and varies depending upon which rule fires. The advantages of using direct database updates are simplicity and flexibility, but at a cost of having to include field identifiers. Fortunately, this overhead can be significantly reduced by enumerating database schema entities and using surrogate keys.

In the ARL prototype, a unit of exchange that includes one atomic event is called a Fact Exchange, or FEX. From a networking perspective, a FEX must be delivered reliably or it will be retransmitted; any uncorrectable bit errors will require the entire FEX to be resent.

A FEX is a variable entity that is determined in the "what part" of the rule. The composition of a FEX is determined dynamically and can depend upon any value(s) stored in the database, to include networking parameters such as average delay, derived bit error rates, etc. For example, suppose that a geographic area (such as a "No Fire Area") is stored as a polygon object that is composed of point objects. In other words, each point and the polygon are a separate database entity, as is illustrated in Figure 4. If a new area is entered into a local database (e.g., No Fire Area 42 as four database objects), a trigger may fire, causing this information to be replicated to other databases.

The actual exchange of information may be arranged in any number of ways as defined by the distribution rule. For example, all four database updates may be sent in a single FEX, each may be sent as its own FEX, or some combination in between these extremes; see Figure 5. In either case, it makes little difference to the database (although for simplicity, one would like to include all related database transaction in one FEX because it simplifies recovery). However, to the network there may be a significant difference if the network is unreliable since the number of bits retransmitted due to failures increases with the size of the FEX. When the transmission network is the bottleneck (as is the case being addressed here), one would want to dynamically adjust packet parameters (such as size and retransmission strategy) to optimize network performance. Thus FEX size may be dictated by the lower layers of the protocols (e.g., the datalink layer) rather than

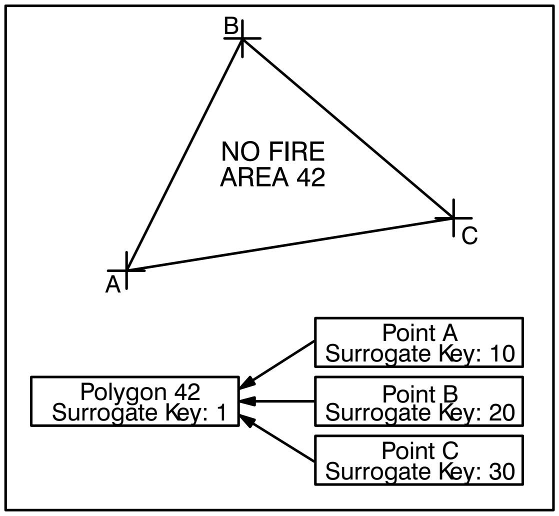


Figure 4: A Geographic Area

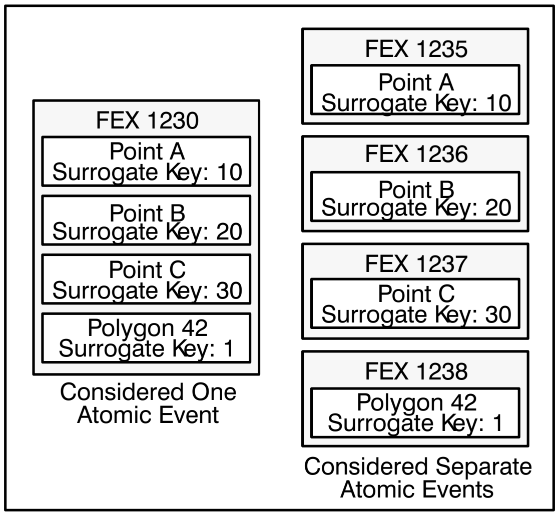


Figure 5: Two Options for Atomic Events

the replication mechanism.[2] This is a first example of the kind of flexibility offered by applying casual, or promiscuous, techniques to the data replication task to accomplish the second objective (to be efficient in sending information with special care not to waste bandwidth).

3.3 End–to–End Reliability

Just as the ''what part'' of a rule identifies which information is to be exchanged within a FEX, the ''where part'' identifies the host destinations of a FEX. The destinations may be identified directly in the rule or indirectly via a database reference. This latter capability allows one to use a single rule for many general cases. For example, one may want to return data to a host that is named in a database transaction or to the host that generated the transaction that caused the rule to fire. Both these cases happen frequently in military operations.

Currently, both reliable and unreliable FEX transfers are supported. Reliable transmissions may be either unicast or multicast and are invoked whenever host addresses are named in the rule (either directly or indirectly). Because the communication environment currently targeted is a single–hop, broadcast network (e.g., either a combat net radio or local area network [LAN]), multicast transmissions simply require more bookkeeping at the Transport Layer [IS 7498].[3] Default transmission parameters for the hosts may be obtained from either configuration files or the database. Typical examples are the host Internet address and network assignment, the number of retransmissions to be attempted for failures, and initial retransmission time–out values. Aliases, such as ''Boss,'' ''Siblings,'' ''Subordinates,'' ''Adjacent,'' etc. are also available. (These are not group addresses, but simply aliases for lists of addresses.)

Unreliable transmission requires no acknowledgment and is referred to as ''broadcast.'' The usefulness of this mode of transmission is controversial but has found frequent use in the ARL prototype because of the transient nature of much of the low–echelon data. Since broadcast involves a communications channel instead of hosts, the channel on which the broadcast is to occur must be identified within a rule. Data broadcast is an excellent example of a ''promiscuous'' replication mechanism because it implies that some data are worthy of transmission but not retransmission. In other words, some data may be considered helpful but not essential, or they are so short–lived that retransmission is pointless.

Because of the ''overhearing'' feature (i.e., all FEXs are passed up from the network layer regardless of addressing), unicast and multicast are equivalent to broadcast for those hosts not listed in the address list. Consequently, if a rule already has a reliable destination listed, then it is redundant to include a broadcast command on the same communications channel since all hosts on that channel will receive the transmission anyway. In a promiscuous replication environment, the only purpose of addressing is to identify those hosts that must acknowledge receipt of the information.

Also linked with the destination addresses is urgency information that is represented by *priority* and *staleness* parameters. Priority is interpreted as expected (i.e., precedence) while staleness refers to the interval of time that the data remains useful and is normally expressed using time. During concept development, there was significant debate as to whether these parameters should be included in the ''what–part'' or the ''where–part'' of the rule syntax. It was decided that, in most cases, urgency was a function of a particular recipient rather than an inherent property of the information itself. Consequently, priority and staleness are associated with the destination of the information. The use of these parameters is discussed in the next section.

4 Transport Layer Protocol Features

Maintaining end–to–end reliability is a significant challenge in an environment where connectivity is intermittent, congestion is high, and failures can reach 50% and greater. Under these conditions, queues grow quickly and retransmission adds to the congestion. One cannot allow the transmission mechanisms to become mired so that information flow is effectively halted. Instead, one must continually re–evaluate which information is the most important and periodically purge the queues of less important updates. This continuous re–evaluation is a key characteristic of a resilient replication mechanism.

2 A FEX is typically much smaller than a data link layer packet. Modern tactical protocols (like 188–220A) support concatenation of upper layer protocol data units so that several FEXs, perhaps all destined for different hosts, could be concatenated and transmitted together in one datalink layer packet.

3 The acknowledgment scheme used is the same as that for ''uncoupled acknowledgments'' in 188–220A [1995].

4.1 Just–in–Time Packet Construction

Just–in–time (JIT) packet construction means that the selection of database updates to be transmitted is postponed until the very last possible moment. It recognizes that in tactical environments, network access may incur significant delays. This feature is not necessary with high-capacity networks because delays are not severe. But in low–bandwidth conditions, where outgoing queues are expected to be large, it is necessary to ensure that the most important information continues to be sent first, even during intervals of high network congestion.

JIT packet construction would normally be implemented at the transport layer of the OSI Reference Model. The idea is that the transport layer protocol maintains a multidimensional, priority queue of database transactions (received from the upper layer protocol [ULP]) awaiting transmission. When data are present in the queue, the transport layer notifies the lower layer protocol (LLP) that it has something to send and then waits for a return signal that the LLP is ready to accept the data. During the wait, more updates may arrive, but since the outgoing packet has not yet been built, there is no penalty for arriving late. When the ‘‘ready’’ signal is received from the LLP, the transport layer selects the updates to be sent and passes them down to the LLP; see Figure 6.

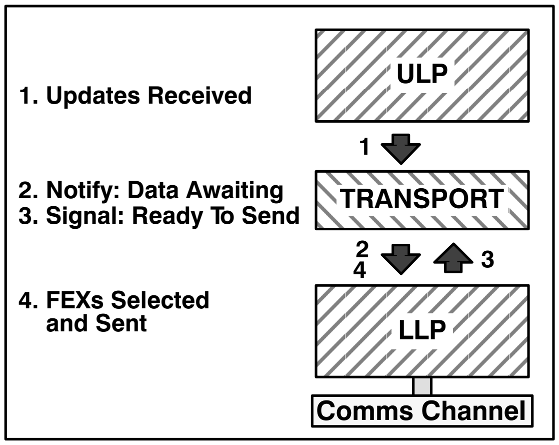


Figure 6: Just–in–Time Packet Construction

The goal is to have the LLP postpone its signal as long as timing allows. Metaphorically, this equates to a railway station waiting for a train to arrive before selecting passengers for boarding. First–come, first–served is no longer in effect. As long as one arrives before the train, he/she will be included in the selection process based on some measure of his/her precedence and not on order of arrival. The goal is to prevent higher precedence passengers from waiting behind lower precedence passengers that arrived earlier.

Precedence may be measured in many ways. In the previous section, urgency was defined in terms of data *priority* and *staleness.* Both of these values would be included as part of the distribution rule and would be passed down from ULP. An update becomes stale when its staleness timer expires; i.e., it has been in the queue longer than its staleness time. Under this scheme, one may remove stale updates from the queue so that they don’t block fresher updates behind them. Similarly, the remaining time of the staleness timer may also be used to calculate transmission precedence.

Most networks have a maximum packet size that is allowed at the various layers of the protocol. This value is typically defined at the Datalink layer and is called a Maximum Transmission Unit (MTU) in the Internet [MIL STD 1777, 1985]. To implement JIT, the transport layer must know the MTU of each network to which it is connected. If the network is congested, there may be many more FEXs pending than can fit into one MTU. Therefore, once the ‘‘ready’’ signal is received from the LLP, the transport layer must not select more FEXs than will fit into the MTU. This calculation must also take into account the amount of overhead that will be added by the LLP. It is important that the transport layer not exceed the MTU, for if the LLP fragments the packet, the purpose of JIT is defeated as information is once again placed in first–in, first–out (FIFO) queues.[4,5]

As previously mentioned, the closer one can coincide packet construction with packet transmission, and the better one can ensure that the most urgent information will get transmitted first. Ideally, for a carrier–sensed, multi–access (CSMA) protocol like those commonly used for tactical environments, this means at least waiting until the transmission slot is selected at the datalink layer. For example, once notified that there are data to send, the new Military Standard 188–220A Tactical Protocol [188–220A, 1996] computes a time delay to

4 For IP, the ‘‘Do Not Fragment’’ option may also be set.

5 One of the tenets of overhearing is that each overheard packet should include usable information. Arbitrary fragmentation will defeat this property.

determine which time slot will be used to attempt to access the network. This delay could be passed up to the Transport Layer so that a back-off time can be computed to determine when to construct the packet and send it to the LLP so that it arrives just moments before the time slot is reached. If the slot is busy, a ‘‘cancel’’ signal could be sent up to the Transport Layer followed by a new slot time, and so on. If timing were perfect, the outgoing packet would be effectively sent by the transport layer directly to the communications channel.

In an ARL prototype implementation, called the Fact Exchange Protocol (FEP) [Kaste, 1990], the active database rules submit database transactions to the FEP for transmission. (The FEP views the transactions as arbitrary chunks of data.) Each submission includes a priority, staleness time and other network-related parameters. The FEP encapsulates each submission into a Fact Exchange (FEX), assigns it a unique tag (called a Fact Exchange ID), and returns that tag back to the ULP for future reference (e.g., to track the progress of its submissions). The FEP places the FEXs in its priority queue and uses JIT packet construction to transmit the FEXs to other hosts. The selection of FEXs for inclusion into outgoing packets is based upon their priority, staleness, time-out status, queue location, and other criteria. If the FEX is successfully transmitted before the staleness time expires, the FEP notifies the upper layer using the associated FEX ID. Alternatively, the Fact Exchange may fail. If this occurs, the FEP notifies the upper layer and provides the reason for failure.

4.2 Fact Exchange Failures

There are three possible causes for a fact exchange (FEX) failure: network congestion, loss of network connectivity, or destination host failure.[6] These failures manifest themselves as two primary types of FEX failures: a staleness failure due to the expiration of a staleness timer, or a retransmission failure due to exceeding the maximum number of retransmissions.

There are two circumstances under which a staleness failure occurs. The first is the normal situation in which during the transmission and retransmission process the timer simply expired. In this case the number of transmissions will be between one and the maximum number allowed. Note that in broadcast cases, a successful FEX is one that was successfully transmitted once (there are no acknowledgments). The second circumstance is the special situation in which the FEX was never even transmitted once. In this case, either the net is so congested that the datalink layer never gained access within the staleness period (global congestion), or the FEXs priority was not high enough to get it included in any outgoing packets (local congestion). It is important to distinguish between these types of failures because different recovery procedures may be followed.

The second type of failure is a retransmission failure. In this case, the FEX is included in an outgoing packet, but an acknowledgment was not received within the retransmission time–out (RTO) period. This transmission process is then repeated up to a maximum number of attempts. When that number is exceeded, a failure is identified. The reason for this type of failure is loss of connectivity, failure of the destination host, or the RTO parameter being set too small. Figure 7 summarizes the type of FEX failures and the associated reason.[7] When a FEX failure occurs, one of these failures is identified and is returned to the ULP to facilitate further investigation in an attempt to determine the underlying situation and recommend a recovery method.

FAILURE TYPE	REASON
Staleness & # tries = 0	Congestion
Staleness & $1 \leq$ # tries $\leq$ Max	Congestion, or Connection, or Host Failure
# Tries > Max	Connection or Host Failure

Figure 7: Reasons for Fact Exchange Failures

5 Failure and Recovery

Just as the transport layer is tracking the success of fact exchanges, the replication mechanism, con-

6 Host failure is often permanent in the tactical environment.

7 Another type of failure is when transmission windows are exceeded. This is when there are too many unacknowledged fact exchanges pending to a particular host or channel. In other words, acknowledgments are not being received. These are handled in the same manner as retransmission failures.

trolled by the active database, is tracking the success of rules. The next step, failure recovery in a resilient system, is a current research topic, and this section discusses some of the consideration to date. An interesting and difficult task is automating the response to failures at the trigger (or rule) level. Recall that the firing of a trigger may invoke a replication command that can generate several database updates, each which is encapsulated as a fact exchange (FEX) by the transport layer protocol. As failures occur, something must be done about them. The *robust* approach would be to keep trying by using another network path, more forward error correction, more power, etc. The *resilient* approach is to determine whether or not there is any value added in further attempts that may potentially increase the congestion and acerbate communication problems. There is a wide range of options: at one extreme, one may give up; at the other extreme, one may continue to retransmit the transaction.

The answer to this quandary usually ends up as the ubiquitous ‘‘it depends on the situation.’’ Fortunately, the strength of the model–based approach is that any information included in the data model (i.e., resident in the database) is available to assist in determining a reaction to failure.

At the rule level, there are at least two considerations for failures. The first is tactical significance. For example, in most cases the failure of a single position location update will not cause a significant synchronization problem, while missing several updates, especially in sequence, may produce a significant problem. However, if a single position update signifies the reaching of a particular goal, such as an intermediate objective, then it may have a significant impact on synchronization. So how one responds to a failure may depend on the event that caused the trigger.

Just as with the fact exchanges, the general question ‘‘what is a failure’’ must be answered for rules. A second consideration of failure is transmission completeness. A replication action is considered ‘‘completely successful’’ if all associated fact exchanges are ‘‘completely delivered.’’ In the example provided in Figures 4 and 5, a geographic area is discussed. Suppose everything except one of the points of a polygon is successfully delivered. The rule will not be successful because all the associated fact exchanges were not completely delivered.[8] Even more subtle, suppose a FEX is multicast and only three of four recipients acknowledge it. Is that an error? The answer to most of these questions is that ‘‘it depends on the situation.’’

One approach being examined is adding an ‘‘else’’ part to the (‘‘if’’ criteria and ‘‘then’’ action) rules that describes what to do if the action is not completely successful. The first task is to determined why the transmission failed so that the chances of a successful retransmission can be calculated. For example, communications connectivity information deduced by passively monitoring the network can be used to determine is a host is dead or just disconnected, see Figure 8. If Host A has a series of failed transmissions to Host B, but it can ‘‘hear’’ Host C acknowledging Host B, then it knows that Host B is active and may just be temporarily masked (e.g., due to geographic obstruction). Further, Host A may know that Host B is heading toward it so that connectivity is likely to improve in the near term. In this case, Host A may wait and retransmit the information in the near future or use Host C as a relay. However, if Host A can hear Host B successfully transmitting to an other host (but not to Host A) then perhaps Host A has a reception problem.

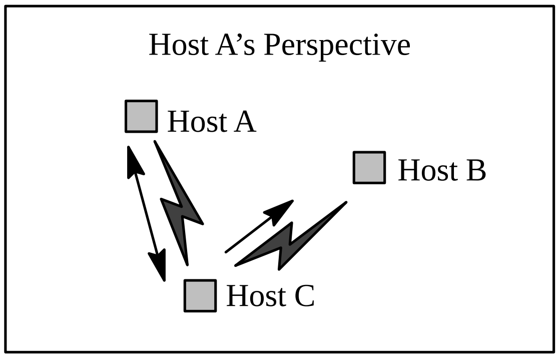


Figure 8: Additional Information Accumulated Via Passive Monitoring of the Network

Thus, further criteria may be included in the ‘‘else’’ part of the rule that peers into the database to collect more details of the failure. In some cases, the only retort may be user intervention. But the goal is to restrict this to ‘‘serious’’ cases. In the future, more

8 This example shows the advantage of including all associated database updates into one fact exchange when possible.

sophisticated capabilities may be included to automate this task, such as mediators [Wiederhold, 1992] or agent systems [Bailey *et al*., 1995].

Another aspect of this problem is to attempt to reduce failures before they occur by restricting the need to communicate (this is the third objective: to replace missing information with predictions based upon *a priori* information stored in the local model). One ongoing ARL project is called ‘‘Objective Sequencing’’ where units exchange a sequence of planned geographic objectives along with thresholds that identify synchronization constraints. Prediction is then used to compare actual progress with expected progress. Thus, information only needs to be exchanged when the actual progress deviates from the expected progress by the threshold. This can significantly reduce the amount of information transmitted by restricting the number of conditions and events that cause rules to fire.

6 Conclusion

Under a model–based paradigm, every unit in the force maintains a local model of the battlefield in its local computing environment. By maintaining communication statistics as part of the model, the synchronization criteria can be a function of the derived communication performance. Thus, as available bandwidth varies, so can the synchronization requirements, and this information can be accessed by the data replication mechanism to assist in selecting an appropriate response to failures. The result is a realistic, resilient data replication approach that allows the synchronization criteria to vary constantly in an attempt to provide a continuous ‘‘best effort’’ rather than a discontinuous guaranteed result. In other words, ‘‘roll with the small punches’’ that may not have a significant impact on the synchronization of the battle.

This goal can be automated by using active database or agent systems. But to accomplish this, one has to modify the strict view of database consistency adhered to by commercial systems. The underlying principle of resilient data replication is to continue to operate, perhaps with or without any significant reduced capabilities, even during frequent, intermittent communication outages. Information generators may have to throw away outgoing transactions that have become stale while monitoring the situation for continuous outages or extreme delays. Information users may have to predict or estimate current values during failures until new information arrives.

In summary, this paper has focused on a view of data replication that deviates from standard commercial practices. This is important in cases like low echelon battle command when, during the ‘‘Fog–Of–War,’’ less accurate but timely information may be more important than exact information. Similarly, robust and resilient methods represent different approaches to solving this problem. To provide resilient battle command systems, one must base the design on the weakest links, which in most cases (especially at the lower ‘‘fighting echelons’’) is the communications systems. In cases of limited bandwidth, one may have to sacrifice an exact (and often not required) audit trail for rapid response of selective data, but this is traditionally considered to be a gross violation of data integrity. Ultimately, there is ‘‘No Free Lunch.’’

References

[188–220A, 1996] Military Standard – Interoperability Standard for Digital Message Transfer Device Subsystems (MIL–STD–188–220A), 27 July 96.

[VMF, 1996] Variable Message Format (VMF) Technical Interface Design Plan For Task Force XXI, Reissue 1, Change 2, 26 January 1996.

[Bailey *et al*., 1995] James Bailey, Michael Georgeff, David Kemp, David Kinny, and Kotagiri Ramamohanarao. Active Databases and Agent Systems – A Comparison. *Lecture Notes in Computer Science: Rules in Database Systems*. Springer–Verlag, Berlin, 1995.

[Chamberlain, 1995] Sam Chamberlain. Model–Based Battle Command: A Paradigm Whose Time Has Come. In *Proceedings of the First International Symposium on Command and Control Research and Technology*, pages 31–38, Washington DC, June 1995. International Symposium on Command and Control Research and Technology.

[Cohen, 1989] Donald Cohen. Compiling Complex Database Transition Triggers. In *Proceedings of the ACM SIGMOD*, pages 225–234, Portland, OR, June 1989. International Conference on Management of Data.

[Davis and Ginn, 1995] Michael Davis and Terry Ginn. Experiences with Replicated Database Systems for Command and Control. In *Proceedings of the First International Symposium on Command and Control Research and Technology*, pages 63–73, Washington DC, June 1995. International Symposium on Command and Control Research and Technology.

[Dayal *et al.*, 1995] Umeshwar Dayal, Eric Hanson, and Jennifer Widom. Active Database Management Systems. *Modern Database System: The Object Model, Interoperability, and Beyond.* ACM Press and Addison Wesley, Reading, MA, 1995.

[Hansen and Widom, 1992] E.N. Hansen and J. Widom. An Overview of Production Rules in Database Systems, IBM Research Report RJ9023, IBM Research Division, 1992.

[IS 7498] Information Processing Systems — Open Systems Interconnection: Basic Reference Model, ISO International Standard 7498.

[Kameny, 1995] Iris M. Kameny. An Approach to Replicated Databases for Robust Command and Control. Rand Technical Report RAND/MR–669–A/ARPA, Rand Arroya Center, October 1995.

[Kaste, 1990] V. Kaste. The Fact Exchange Protocol. A Tactical Communications Protocol. BRL Memorandum Report BRL–MR–3856, August 1990.

[McCarthy and Dayal, 1989] Dennis McCarthy and Umeshwar Dayal. An Architecture of an Active Database Management System. In *Proceedings of the ACM SIGMOD*, pages 215–224, Portland, OR, June 1989. International Conference on Management of Data.

[MIL STD 1777, 1985] US Military Standard 1777, ‘‘Internet Protocol,’’ DDN Protocol Handbook, Vol 1, December 1985.

[Wiederhold, 1992] Geo Wiederhold. Mediators in the Architecture of Future Information Systems. IEEE Computer, 25(3): pages 38–49, March 1992.